Language Processors – Assignment 2

This assignment is for **10 Marks.** It is based on preprocessing C language program. The input would be a C language program and output would be a valid C language program. Both the programs should be compiling and producing same output for same input.

You will need to form a group of 2 or (at a maximum) 3 students. The assignment would be based on submission followed by viva.

There would be 4 problems and each group would be assigned one problem randomly by your TA instructor.

The submission date and time is Sep 16, 12 Noon. The group would have to be present in the lab timings on any day of the week of Sep 16 (except Thursday). Late submission is accepted till Oct 14, 12 Noon where the group would be evaluated for 60% of the maximum marks.

Problem 1: Convert "for-loop"/"do-while loop" to "while loop" without changing the meaning of the program. There may be loops inside loops and so on. Marks are reserved for properly converting "break" and "continue" statements.

Problem 2: Convert "switch-statement" to "if-else-statements" without changing the meaning of the program. There may be a switch inside a switch and so on. Marks are reserved for properly converting "default" clause and "break" statement.

Problem 3: Write code to process macros. A macro is to be substituted as inline code.

For Example:

#define f(x) x*x

If anywhere f(x) appears in any function then it is replaced by the right side of the macro substituting for the argument 'x'. For example, f(3) would be replaced with 3*3. Remember that a macro can call another macro.

Also remember that there can be self-referential macros or mutually-recursive macros, which need to be flagged as errors.

Another example can be

```
typedef struct {
            #define DECLARE_MY_VAR(member, type) type member;
            DECLARE_MY_VAR(i, int);
   } myStruct;
```

Which would get converted to:

```
typedef struct {
            int i;
   } myStruct;
```

For yet another example, consider the following code snippet:

```
            #define MY_INT 137
            int x = MY_INT; // MY_INT is replaced
            #undef MY_INT;
            int MY_INT = 42; // MY_INT not replaced
```

The preprocessor will rewrite this code as

```
            int x = 137;
            int MY_INT = 42;
```

<u>Problem 4:</u> Extend the C language to support power (^) operator. (This is the only case where input C language code is not compilable, but output C language code should compile)

Extend the C language by parsing the power (^) operator and replacing all the occurrences of (^) by calling a function for returning power of a number. (You can even use library function pow() for the same, it is defined in math.h).

For example: 2^3 should result in either calling your function or calling pow(2,3).

You should be able to handle floating point numbers as well as –ve numbers / variables.

You should be able to handle brackets "()" as well.

For example: ((2+3)*x)^(y+2)^z should translate properly to pow(pow((2+3)*x, y+2), z).